A BUCKET-INTERLEAVING MULTIPLEXER FOR EFFICIENT NEAR-ON-DEMAND STREAMING TO RESOURCE-CONSTRAINED CLIENTS

Yitzhak Birk and Yair Wiener

Technion – Israel Institute of Technology
birk@ee, wyair@tx.technion.ac.il

ABSTRACT

Bandwidth-optimal open-loop near-on-demand streaming (NVOD) entails the optimal assignment of transmission rates to a large number of program segments. These are transmitted concurrently and repetitively at their assigned rates. At any given time in its viewing of the program, a client must record data belonging to a contiguous subsequence of the segments for subsequent display. Limited client recording rates and storage capacity affect the rate assignments. In practice, the concurrent streams must be time-multiplexed onto a single channel, and efficient operation of the client disk drive used to store received data until its viewing prevents fine-grain multiplexing. The bucket-interleaving multiplexing scheme presented in this paper guarantees that each segment is repeated in its entirety within any contiguous time interval of the appropriate length, and reduces the required client "rate-smoothing" RAM buffer by two orders of magnitude relative to pure earliest-deadline-first multiplexing.

1. INTRODUCTION

"Near On-Demand" (NOD) streaming offers to an unlimited number of concurrent (albeit not simultaneous) "clients" viewing flexibility that closely approximates "On-Demand" service, but does so at a very low amortized cost to the server and network. E.g., a client may be guaranteed commencement of service within 30 seconds of request along with the ability to pause at any time and resume instantaneously. This is particularly attractive to service providers as a means of offering "hot" titles. Of special benefit are the *open loop*, or "broadcast and select" schemes, whereby the server transmits data independently of viewer actions, and each client selects the relevant subset of the received data. These schemes feature perfect scalability. Also, their broadcast-and-select nature makes them particularly attractive for broadcast-oriented infrastructures such as cable and satellites. The core service can even operate with one-way communication.

A simple open-loop scheme, applied to a 100-minute movie and a 30sec viewing-commencement delay, entails the repetitive transmission 200 video streams, staggered at 30-second intervals. While this scheme requires a fixed data rate for an unlimited number of viewers, the number of streams is often prohibitive. Instead, virtually all schemes (e.g., [1][2][3][4][5][6]) are based on the observation that, for later movie segments, more time is available from viewing request until the actual viewing. They also rely on the availability of storage space in every client machine, which can be used to record prematurely-received program segments for subsequent display. Thus, the need to transmit each segment **at the time** that it is displayed by any given client is replaced with the need for the client to have received it **by that time**.

The lowest possible aggregate transmission rate is attained by partitioning a program into sufficiently small segments and transmitting them such that each segment is transmitted in its entirety within any contiguous time interval equal to the time from a client's request to view a program until the viewing of that segment. This reduces the aggregate mean transmission rate (in units of the program's original data rate) from the ratio between program duration and the viewing-commencement delay to the natural logarithm of that ratio [1][6]. (From 200 to 5.3 times the video rate in the above example).

While the above transmission rate is very low, a client must be able to store 37% of the movie [6] and to record at a rate equal to the aggregate transmission rate. This has given rise to research into efficient ways of reducing client resource requirements at the cost of an increase in aggregate transmission rate (e.g., [2][3][6]). Many of these schemes call for the concurrent repetitive transmission of all program segments at various rates, said rates specified by the particular scheme.

Unfortunately, it is impossible to literally transmit all segments concurrently. Instead, the transmissions must be time-multiplexed onto a single transmission channel. Moreover, in order to operate the client disk efficiently without requiring huge RAM buffers, the interleaving of data chunks from different segments must be carried out at a fairly coarse granularity, at least several tens of kilobytes. In this paper, we present a dynamic time-multiplexing algorithm for emulating the concurrent transmission of the numerous segments.

Unlike previous work on periodic broadcast schedules for sets of items, such as [7], whose only concern is with the time until any given item is received by a waiting client, we must also consider the client's limited recording rate.

0-7803-7304-9/02/$17.00 C2002 IEEE

Coarse-grain emulation of concurrent transmission of all segments at the respective rates causes dramatic variations in the required recording rate over fairly large time windows. This, in turn, requires substantial RAM buffering in the client for "rate smoothing". Mitigating this problem requires joint consideration of the different segments when designing the interleaving schedule. The problem is complicated even further by the fact that a client may begin its viewing and recording at an arbitrary time and, as its viewing progresses, the sequence of segments whose data it must record forms a "sliding window". Moreover, we must give hard guarantees rather than merely mean values. The main contribution of this paper is a multiplexing scheme that succeeds in jointly addressing all the issues, and provides hard guarantees, all this with a moderate and predictable transmission-rate overhead.

The remainder of the paper is organized as follows. In section 2, we state the formal requirements, present our scheme, and analyze its performance. In section 3, we briefly address a fragmentation problem that arises. Section 4 presents a numerical example, and Section 5 offers concluding remarks.

2. MULTIPLEXING ALGORITHM

Given movie parameters (e.g., video rate as a function of time), available client storage and maximum client recording rate, a rate-assignment algorithm breaks the movie into (many) segments and assigns a transmission rate to each. As shown in [6], the optimal transmission-rate assignments and the associated client action that minimize client resource consumption are such that 1) the client commences to record transmitted chunks of any given program segment no earlier than it starts recording chunks of any earlier segment, and 2) the recording interval of a segment is contiguous, ending just before its viewing commences. From the above, it follows that the segments whose chunks should be recorded by a client at any given time form a contiguous subsequence of the movie, and this subsequence is a "sliding window" whose boundaries shift towards later parts of the movie as viewing time progresses. While the multiplexing scheme presented below is independent of the specific rate assignments, it does assume the above properties.

The rate-assignment algorithm is assumed to have chosen segment sizes that are integer multiples of some fixed chunk size. Because there are typically at least several thousand chunks per movie, this is a negligible restriction. Chunks are taken as the unit of scheduling and of buffer-memory management.

The multiplexing algorithm must 1) ***preserve correctness*** (Every segment must be transmitted in its entirety in any time interval of sufficient length), 2) ***minimize*** the required (maximum over viewing time and client arrival times) ***client RAM "rate-smoothing" buffer size***, and 3) minimize the transmission overhead required for achieving 1) and 2).

2.1. The algorithms

Consider a sequence of segments, each with an assigned transmission rate. (The range of segments that any given client must record at any given time follows from the given rate assignments.) Each segment comprises several fixed-size chunks (same chunk size for all segments). We begin by addressing the burstiness in recording rate. Consider time slots equal to the transmission time of a chunk at the aggregate transmission rate. The general idea is to guarantee that for any number of consecutive time slots greater than some (small) specified number, the fraction of chunks that must be recorded by any given client will never exceed (maximum recording rate / aggregate transmission rate) by more than a small specified margin. This requirement is directly related to the required amount of per-client "rate smoothing" buffer space. We next address this by limiting the length of a transmitted burst of chunks that may all have to be recorded by some client.

Algorithm 1. Spacing and coarse multiplexing

1. Partition the aggregate transmission rate equally into N_b "buckets";
2. Assign the segments, in movie order, to the lowest-number bucket that is not full, and reduce the available bandwidth of this bucket by the assigned segment's transmission rate;
3. Transmit chunks from the buckets in round robin bucket order, one chunk per bucket per round, with buckets selected per the following formula:

$$b[i] = \begin{cases} c \cdot i \mod (N_b - 1) & 0 \le i < N_b - 1 \\ N_b - 1 & i = N_b - 1 \end{cases},$$

where *b[i]* denotes the bucket whose chunk is transmitted in time slot *i*, and *c* denotes the smallest integer factor of N_b that is greater than or equal to N_{cb}, the maximum (over time) number of buckets whose data a client must record. (N_{cb} is approximately equal to the aggregate transmission rate over the maximum recording rate as determined by the rate-assignment algorithm.)

Remarks.

- By placing same-segment chunks as well as those of adjacent segments in the same bucket (step 2) and serving the buckets in a round-robin manner, we achieve uniform spreading of the transmission times of chunks that must be recorded by a client at any given time. The specific order in which buckets are served furthermore extends this spreading to chunks that reside in adjacent buckets and are thus also likely to have to be recorded by a client during similar time intervals.
- Buckets may remain partly filled. This "fragmentation" is not serious in practice, but will be addressed later.

Having decided how to assign segments to buckets and how to allocate time slots among the buckets, we next turn our attention to intra-bucket scheduling, i.e., to the allocation of the time slots among the segments in a given bucket. This should guarantee correctness.

Algorithm 2: intra-bucket multiplexing

```
RequestChunkTransmission ()
1. for (i=1; i<=Nsegments; i++){
2. period[i]= sizeOfChunk/transmissionRate[i];
3. at period[i] intervals, insert a
   transmitChunkOfSegment[i] request into the transmission
   queue of the bucket to which segment i was assigned;
4. deadline(transmission request)=currentTime+period[i]
5. }

TransmitChunks ()
6. TimeSlot: = ChunkSize / AggregateTransmissionRate;
7. Every (TimeSlot) do:
8. Select next bucket (cyclic order);
9. From the selected bucket, transmit the next chunk (cyclic
   order) of the segment with the earliest deadline;
```

We next state the main properties of our multiplexing scheme. Proofs and derivations are omitted for lack of space.

Lemma 1: A client must be capable of recording (from buffer to storage) at a rate of $R_r = \frac{N_{cb}}{N_b} \cdot R_t$, where R_t is the aggregate transmission rate. □

Theorem 2: A client buffer size that is guaranteed to smooth the recording rate over one transmission round, S_b^{min}, is:

$$S_b^{\min} \le \left\lceil \frac{(N_b+2)^2}{8N_b} \right\rceil \cdot S_c.$$

where S_c is the (fixed) size of a chunk. □

This bound was obtained by taking the maximum over all values of c and N_{cb}. In any specific case, the required buffer space may be smaller.

Given R_t', the aggregate transmission rate as determined by the rate-assignment algorithm (the sum of the rates assigned to the segments), let us next derive the aggregate transmission rate of the multiplexer, R_t, that is required in order to guarantee correctness. The difficulty stems from the multiplexing and the resulting loosely controlled inter-chunk periods for any given segment. By adopting and adapting a result of [8] and [9] for an LTRB (least time to reach bound) policy, we state the following theorem.

Theorem 3: The actual transmission rate assigned to a segment must be higher than the rate originally assigned to it by the "continuous" rate assignment algorithm by a factor of $(N_c+1)/N_c$. Therefore, $R_t=R_t'(N_c+1)/N_c$. □

Let us next consider the required client recording rate. The increase in required recording rate beyond R_r', the originally required rate, is due to 1) the increase in transmission rate beyond the original one, and 2) the coarse granularity of transmission-rate partitioning among the buckets. Referring to the latter, even if a bucket contains only one segment that a client needs to record, we must assume that a chunk of that segment will be transmitted in any given round. If the sum of the transmission rates of the segments that a client must record at a given time slightly exceeds k times the transmission rate allocated to a single bucket, these segments may reside in up to $k+2$ consecutive buckets (of which they will fully occupy k). Therefore,

Theorem 4: The maximum (over viewing time and arrival time) client recording rate is tightly bounded from above by

$$R_r^{\max} = \frac{\left\lceil \frac{R_r'}{R_t'} N_b \right\rceil + 1}{N_b} \cdot \left(1+\frac{1}{N_c}\right) \cdot R_t' \le \left(1+\frac{1}{N_c}\right) R_r' + \frac{2}{N_b} \cdot \left(1+\frac{1}{N_c}\right) R_t'.$$

3. INTERNAL FRAGMENTATION

The constraint whereby any given segment must be assigned to a specific single bucket results in wasted space (transmission-rate allocations) in buckets. The severity of this "internal fragmentation" problem depends on the number of buckets, segment size, etc. We have experimented with two approaches, both of which solve the problem with negligible (sub-1%) transmission-rate overhead: ***Hole filling***, whereby last-bucket segments are moved to the first buckets, and ***Segment-size tuning.*** (Hole filling works because 1) a client records very few buckets towards the end, so adding one or two is not a problem, 2) the first and last bucket are cyclically consecutive, 3) the last segments have the lowest data rates and are thus the best "fillers", and 4) the first buckets contain the highest rate segments and thus have the greatest fragmentation problems.)

4. AN EXAMPLE

In this section, we illustrate the benefits of our scheme with an example. Consider a 100-minute movie with a video rate of 512KB/s. The permissible delay from request to viewing commencement is 30sec. We use one-second segments (512 KB). Client constraints: R_r=1000 KB/s; storage capacity (disk) of 400,000 KB. Based on these constraints, a transmission-rate assignment algorithm [6] yielded an aggregate transmission rate R_t' = 3,718 KB/s.

Fig. 1 depicts the resulting client recording rate and storage consumption (not RAM buffer) versus viewing time. Note the decrease in recording rate over time. Also,

note that the client storage is empty at the beginning and at the end.

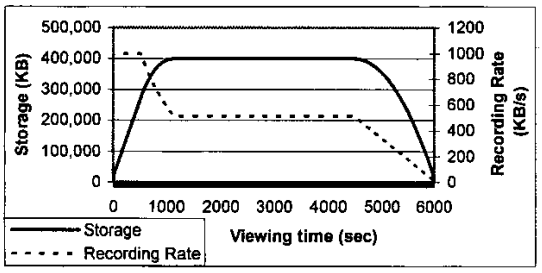

Figure 1: Recording rate and client storage (not buffer) utilization versus viewing time with "ideal" concurrent transmission of all segments at the originally assigned rates.

Next, let us turn to the multiplexing. Table 1 depicts a comparison, based on actual simulation results, between pure earliest-deadline-first (EDF) chunk interleaving and our bucket-based scheme. Two chunk sizes were considered: 32KB and 64KB. The increases in transmission and recording rates (relative to R_t and R_r, respectively) represent the overheads that were derived in this paper. In order to comply with the original constraints, one could simply make them artificially more strict by the derived overhead factors and then execute the rate assignment algorithm (off line) for those new requirements, followed by the multiplexing. Note the tremendous advantage of our scheme over the basic EDF in terms of client buffer requirements.

TABLE I
RESOURCE REQUIREMENTS WITH PURE EARLIEST DEADLINE FIRST (EDF) MULTIPLEXING AND WITH OUR BUCKET-INTERLEAVING (BUCK-INTRLV) SCHEME.

Sched	Chunk Size	R_t [1] <KB/s>	R_r <KB/s>	N_b	Req. client buffer
EDF	32 KB	3958	1326	1	11.4 MB [2]
BUCK-INTRLV	32KB	3958	1326	30	32 KB [3]
EDF	64 KB	4191	1404	1	18.1 MB [2]
BUCK-INTRLV	64KB	4191	1404	30	64 KB [4]

Notes:

1. In order to solve the remaining minor fragmentation problem (after using hole filling), we increased the total transmission rate by a mere 0.2%.
2. With EDF, the buffer requirements varied depending on the client's arrival time, between seven and several tens of megabytes in measurements that we carried out.
3. This result was obtained when the fragmentation problem was not addressed (slightly increasing transmission rate). When using the hole-filling approach, the minimum buffer size was 128 KB.
4. This result was obtained when the fragmentation problem was not addressed (thus requiring a slightly higher transmission rate). When using hole filling, the minimum buffer size was 192 KB.

In practice, buffer sizes should be rounded up to be an integer multiple of chunk size. Also, one should add a single chunk-size to the above results to prevent read/write collisions.

5. CONCLUSIONS

An insightful characterization of the client-recording pattern in efficient open-loop near-on-demand streaming schemes enabled us to devise a simple yet novel stream-interleaving scheme. This scheme dramatically reduces required client buffer size (by two orders of magnitude in our example) relative to earliest-deadline-first scheduling of data chunks. Moreover, the required resources can easily be determined and are guaranteed to suffice.

The interleaving scheme proposed in this paper and its analysis only apply to situations wherein the maximum number of buckets whose chunks must be recorded by any given client in any given round is at most one half of the total number of buckets. This is very common in the case of constrained client recording rate or storage capacity, but may not always be the case. In such situations, non-ideal interleaving can nonetheless still substantially reduce buffer requirements relative to those of pure EDF. A formal extension of our interleaving scheme and its analysis to this range is a topic for further research.

6. REFERENCES

[1] H. C. De Bey, "Program transmission optimisation", United States Patent Number 5,421,031, Mar 1995.

[2] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting", Multimedia Systems, vol. 4, pp. 197-208, 1996.

[3] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems", Proc. IEEE Intl. Conf. on Multimedia Computing and Systems, pp. 118-126, Jun 1996.

[4] L-S. Juhn and L-M. Tseng, "Harmonic broadcasting for video-on-demand service", *IEEE Trans. Broadcasting*, vol. 43, no. 3, pp. 268-271, Sep 1997.

[5] L-S. Juhn and L-M. Tseng, "Staircase data broadcasting and receiving scheme for hot video service", *IEEE Trans. Consum. Elec.*, 43(4), pp. 1110-1117, Nov 1997.

[6] Y. Birk and R. Mondri, "Tailored transmissions for efficient Near-Video-On-Demand service", Proc. IEEE Multimedia Systems (ICMCS), pp. 226-231, May 1999.

[7] M.H. Ammar and J.W. Wong, "The design of teletext broadcat cycles", *Perf. Eval.*, vol. 5, pp.25-242, 1985.

[8] A. Birman, H.R. Gail, S.L. Hantler, Z. Rosberg and M. Sidi, "An Optimal Service Policy for Buffer Systems", *J. ACM*, vol. 42(3), May 1995, pp.641-657.

[9] I. Cidon, I. Gopal, G. Grover and M. Sidi, "Real time packet switching: A performance analysis", IEEE J. Sel. Areas Commun., vol. 6, Dec. 1988, pp. 1576-1586.